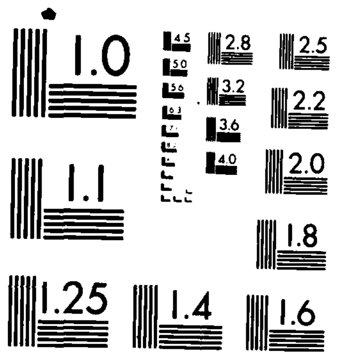
AD-A161 358 DERIVATION OF RANDOMIZED ALGORITHMS(U) HARVARD UNIV CAMBRIDGE MA AIKEN COMPUTATION LAB S RAJASEKARAN ET AL OCT 85 TR-16-85 N00014-80-C-0647 1/1

UNCLASSIFIED F/G 12/1 NL

END

FILMED

DTIC


1.0
4.5
5.0
5.6
6.3
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

(12)

AD-A161 358

DERIVATION OF RANDOMIZED ALGORITHMS

Sanguthevar Rajasekaran
and
John H. Reif

TR-16-85

Harvard University

Center for Research in Computing Technology

Aiken Computation Laboratory
33 Oxford Street
Cambridge, Massachusetts 02138

12

DERIVATION OF RANDOMIZED ALGORITHMS

Sanguthevar Rajasekaran
and
John H. Reif

TR-16-85

unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD A164 358	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* DERIVATION OF RANDOMIZED ALGORITHMS		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER TR-16-85
7. AUTHOR*(s)* Sanguthevar Rajasekaran, John H. Reif		8. CONTRACT OR GRANT NUMBER*(s)* N00014-80-C-0647
9. PERFORMING ORGANIZATION NAME AND ADDRESS Harvard University, Cambridge, MA 02138		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research, 800 North Quincy Street, Arlington, VA 22217		12. REPORT DATE October 1985
		13. NUMBER OF PAGES 23
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* Same as above		15. SECURITY CLASS. *(of this report)*
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

unclassified

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

unclassified

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

randomized algorithms, derivation, parallel algorithms, searching, sorting, selection

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

See reverse side.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

ABSTRACT

This paper surveys a number of efficient randomized algorithms for selection and sorting which we derive from inefficient deterministic specifications. Along with these derivations, we simultaneously derive bounds on the probability distribution of the sequential and parallel time cost of these algorithms.

There are several potential benefits of this work. We develop, for the first time, geneal techniques for deriving randomized algorithms from deterministic specifications. Previous derivations only considered deterministic algorithms. Furthermore, our derivations encompass many known sequential and parallel randomized algorithms for selection and sorting which required separate proofs and probabilistic analysis.

Also, we present a new randomized comparison sorting algorithm that takes $O(\log\log n)$ time and uses $n^{1+\epsilon}$ processors to sort n keys.

n1+epsilon

Derivation of Randomized Algorithms

Sanguthevar Rajasekaran

John H. Reif

Aiken Computing Lab.

Harvard University

October 17, 1985

ABSTRACT

This paper surveys a number of efficient randomized algorithms for selection and sorting which we derive from inefficient deterministic specifications. Along with these derivations, we simultaneously derive bounds on the probability distribution of the sequential and parallel time cost of these algorithms.

There are several potential benefits of this work. We develop, for the first time, geneal techniques for deriving randomized algorithms from deterministic specifications. Previous derivations only considered deterministic algorithms. Furthermore, our derivations encompass many known sequential and parallel randomized algorithms for selection and sorting which required separate proofs and probabilistic analysis.

Also, we *present* a new randomized comparison sorting algorithm that takes $O(\log\log n)$ time and uses $n^{1+\epsilon}$ processors to sort n keys.

1. INTRODUCTION

1.1 Meaning of Derivation

A growing body of computer science literature is concerned with deriving efficient algorithms for given mathematical specifications which would be inefficient to execute as such. For example, [Scherlis 80] derives a family of parsing algorithms and [Reif and Scherlis 84] derive a family of efficient depth first search algorithms for various connectivity problems of graphs and digraphs. The process of deriving algorithms from given specifications could be thought of as restating the specifications in steps, using general algorithm design techniques and problem specific knowledge, such that the specification at any step of derivation reflects an improvement in the efficiency of its execution over the one in the previous step. The advantage of this derivation process is manyfold. These derivations serve as constuctive proofs for the correctness of a family of algorithms. Also, they enable us to understand existing efficient (and possibly complicated) algorithms for the problem at hand better, than merely reading the proofs of their correctness. In this paper we derive randomized algorithms for selection and sorting.

1.2 Randomized Algorithms

A randomized algorithm A defines a mapping from an input domain D to a set of probability density functions over some output domain D'. For each input $x \in D$, $A(x) : D \rightarrow [0,1]$ is a probability density function, where $A(x)(y) \in [0,1]$ is the probability of outputting y given input x. In order for $A(x)$ to represent a probability mass function, we require

$$\sum_{y \in D'} A(x)(y) = 1, \text{ for each } x \in D.$$

A mathematical semantics for randomized algorithms is given in [Kozen 80].

Two different types of randomized algorithms can be found in the literature: 1)those algorithms which always output the correct answer but whose run time is a random variable with a specified mean, and 2)those which output the correct answer with some probability (the probability space being the set of all possible inputs). For example, the randomized sorting algorithm of [Reischuk 81] is of the first type and the primality testing algorithm of [Rabin 76] is of the second type. In general, the use of probabilistic choice in algorithms to randomize them has often lead to great improvements in their efficiency. The randomized algorithms we derive in this paper will be of the first type.

1.3 Deriving Randomized Algorithms

No previous paper has given derivations of randomized algorithms (for any problem). We begin with the mathematical specifications of selection and sorting problems. We'll encode these specifications into *canonical* algorithms. Various algorithms found in the literature for these problems will then be derived as special cases of these canonical algorithms. The former algorithms result from specializing and/or modifying one or more of the steps in the canonical algorithms.

To start with, we derive and analyze a random sampling algorithm for approximating the rank of a key (in a set). This random sampling technique will serve as a building block for the selection and sorting algorithms we derive. We analyze the run time for both the sequential and parallel execution of the derived algorithms.

1.4 An $O(\log\log n)$ Time Sorting Algorithm

Many optimal parallel comparison sorting algorithms are available in the literature. These algorithms are optimal in the sense that the product of time and processor bounds for these algorithms equals the lower bound of the run time for sequential comparison sorting. These processors run in time $O(\log n)$. Some of these algorithms are 1)Reischuk's[Reischuk 81] randomized algorithm, 2)AKS deterministic algorithm[AKS83], 3)Column Sorting algorithm [Leighton 83], 4)FLASH SORT algorithm[Reif and Valiant 83]. The first three algorithms run on the PRAM model whereas the fourth algorithm runs on the fixed connection model. We give a new (non optimal) sorting algorithm that runs in time $O(\log\log n)$ time and which uses $O(n^{1+\epsilon})$ processors, for any $\epsilon>0$.

1.5 Organization of this Paper

In section 2 we define the selection and sorting problems and our parallel comparison tree machine model. In section 3 we give and analyze an algorithm for computing the rank of a key approximately. We also state, in this section, an important result from sampling theory which will be used throughout the rest of the paper. Finally, in sections 4 and 5 we derive and analyze various randomized algorithms for selection and sorting. In section 5 we also give a new comparison sorting algorithm that runs in time $O(\log\log n)$ time.

2. COMPARISON PROBLEMS AND PARALLEL MACHINE MODELS

2.1 Comparison Problems

Let X be a set of N distinct keys. Let $<$ be a total ordering over X. For each key $x \in X$ define

$$\underline{\text{rank}}(x,X) = |\{x' \in X \ / \ x' < x\}| + 1.$$

For each index i, $1 \leq i \leq N$, we define $\underline{\text{select}}(i,X)$ to be that key $x \in X$ such that $i = \text{rank}(x,X)$. Also define

$$\underline{\text{sort}}(X) = (x_1, x_2, \ldots, x_N)$$

where x_i = select(i,X), for i=1,...,N. As defined, these functions are expensive to compute. For example, the sort definition requires N^2 comparisons.

2.2 Parallel Comparison Models

In the *sequential comparison tree* model of [Knuth 73] a single step at a node of the tree consists of a comparison of two keys. The outcome of this comparison takes the execution to a child of this node. The leaves of the tree provide output values. The run time in this model is the number of nodes visited on a given execution. A distinct tree is allowed for each input. In a randomized comparison tree model execution from any node branches to a random child depending on the outcome of a coin tossing.

[Valiant 75] describes a *parallel comparison tree* machine model which is similar to the sequential tree models, except that multiple comparisons between keys are allowed on each step. Thus a comparison tree machine with p processors is allowed a maximum of p comparisons at each node, which are executed simaltaneously. We allow our parallel comparison tree machines to be randomized, with random choice nodes as described above.

2.3 Parallel RAM Models

More refined machine models of computation also take into account storage and arithmetic steps. The sequential random access machine (RAM) described in [Aho, Hopcroft, and Ullman 76] allows a finite number of register cells and also infinite global storage. A single step of the machine consists of an arithmetic operation, a comparison of two keys, reading off the contents of a global cell into a register, or writing the contents of a register into a global memory cell.

The parallel version of RAM proposed by [Fortune and Wylie 78] allows multiple RAMs to be generated from a single original RAM by execution of a *fork* operation. This model, known as PRAM, allows multiple concurrent reads but prohibits concurrent writes. WRAM model, which is a variation of PRAM, permits concurrent reads and concurrent writes. There are three variations of WRAM depending on how the write conflicts are resolved.

The models we employ, in this paper, for various algorithms will be the ones used by the corresponding authors.

3. RANDOM SAMPLING

3.1 An Algorithm for Computing Rank

Let X be a set of N keys with a total ordering $<$ defined on it. Our first goal is to derive an efficient algorithm to approximate rank(x,X), for any key $x \in X$. We require that the output of our randomized algorithm have expectation rank(x,X). The idea will be to sample a subset of size s form X, to compute the rank of x in this sample, and then to infer its rank in X. The actual algorithm is given below.

```
algorithm samplerank$_s$($x$,$X$);
  begin
    Let $S$ be a random subset of $X$ of size $s$;
    return $\left\lceil \left( \frac{N}{s+1} \right) \text{rank}(x, S \cup \{x\}) \right\rceil$
  end;
```

The correctness of the above algorithm is stated in the following

Lemma 3.1 The expected value of samplerank$_s$(x,X) is rank(x,X).

Proof Let the number of elements $\leq x$ in$X$ be k (i.e., rank(x,X) = k). Then, for a random $y \in X$, Prob.$[y \leq x] = \frac{k}{N}$. Therefore, out of s elements chosen randomly from X, Prob.[exactly l of them are $\leq x$] $\leq B\left(l;\, s+1, \frac{k}{N}\right)$ (where B is a binomial distribution). Mean of this binomial distribution is $(s+1)\frac{k}{N}$. Therefore,

$$\text{Mean}\left[\frac{N}{s+1}\,\text{rank}(x, S \cup \{x\})\right] = k = \text{rank}(x, X)$$

$$\text{i.e., Mean }[\text{samplerank}_s(x,X)] = \text{rank}(x,X) \ \square$$

Another way of proving the correctness of our algorithm is to use some well known results from Sampling Theory. We address ourselves to the following problem: "Given a set X and a random $x \in X$. Also, S is a random subset (or sampling) of X. If the number of elements $\leq x$ in S is l, how many elements in X are $\leq x$?" Let r_i= rank(select(i,S), X). In the next subsection we'll obtain the distribution of r_i and show that Mean(r_i) = $i\frac{N}{s+1}$.

3.2 Distribution of r_i

Let $Y_1, Y_2, \ldots, Y_s$ be a random sample from a continuous distribution with density function $f(y)$. If $Y_{(1)} \leq Y_{(2)} \leq \cdots \leq Y_{(s)}$ is the sorted order of the sample, then $Y_{(i)}$ is called the ith order statistics of the sample $Y_1, \ldots, Y_s$. If $f(y) = 1$ in [0,1] and 0 elsewhere, then the density function of the ith order statistics [Wilks 76] is given by

$$f_i(y_{(i)}) = \frac{s!}{(i-1)!(s-i)!} y_{(i)}^{i-1} (1-y_{(i)})^{s-i} ,$$

which is a Beta distribution with parameters $(i, s-i+1)$. Therefore, asymptotically, $\frac{r_i}{N}$ has a Beta distribution with parameters $(i, s-i+1)$, immediately implying that Mean(r_i)=$i\frac{N}{s+1}$ and Var(r_i)=$\frac{i(s-i+1)}{(s+1)^2(s+2)}N^2$. From this Beta distribution we can also obtain the following confidence interval on r_i.

Lemma 3.2 For every α, Prob. $\left(| r_i - i\frac{N}{s+1} | > c\alpha \frac{N}{\sqrt{s}} \sqrt{\log N} \right) < N^{-\alpha}$ for some constant c.

Proof We'll assume $\log N = o(s)$ in the following proof. Also, this proof is given in two cases on the parameters of the Beta distribution.

Case1 Both i and $s-i+1$ are $>\sqrt{s/\log N}$.

Let Z be a random variable with probability density function $f(z)$, mean μ, and variance σ^2. It is a known fact from statistics that if the skewness of Z is 0, then $\frac{Z-\mu}{\sigma}$ can be approximated by the normal variate N(0,1). The skewness of a Beta distribution with parameters (α, β) is 0 when either $\alpha = \beta$ or both α and β are ∞. Therefore, when both i and $s-i+1$ are $>\sqrt{s/\log N}$, asymptotically, the skewness of r_i tends to 0, making the approximation of $\frac{r_i-\mu}{\sigma}$ by the standard normal variate meaningful. And hence,

$$\text{Prob.}\left\{\left|\frac{r_i-\mu}{\sigma}\right| > t\right\} = 2\int_t^\infty \exp(-u^2/2)du$$

$$\leq 2\,\frac{1}{\sqrt{2\pi}}\,\frac{\exp(-t^2/2)}{t}$$

$$\leq b\,\exp(-dt^2) \leq b\,N^{-\alpha}, \text{ for constants } b \text{ and } d$$

$$\text{and } t=\sqrt{\frac{\alpha\log_e N}{d}}.$$

Notice that $\text{Var}(r_i) \leq \frac{N^2}{2s}$. Now, substituting $\text{Mean}(r_i) = i\frac{N}{s+1}$ and $\text{Var}(r_i) \leq \frac{N^2}{2s}$, we get the desired result.

Case2 Either i or $s-i+1$ is $\leq \sqrt{s/\log N}$.

The case $i \leq \sqrt{s/\log N}$ is identical to the case $(s-i+1) \leq \sqrt{s/\log N}$. So, we'll consider only the case $i \leq \sqrt{s/\log N}$.

If $k_1,k_2,\ldots,k_s$ are the elements of the random sampling set S in sorted order, then these elements divide the set X into $(s+1)$ subsets $X_1,X_2,\ldots,X_{s+1}$ where $X_1 = \{x\in X \;/\; x\leq k_1\}$, $X_i = \{x\in X \;/\; k_{i-1}<x\leq k_i\}$, for i=2,...,s and $X_{s+1} = \{x\in X \;/\; x>k_s\}$. In terms of the cardinalities of these subsets, r_i can be expressed as $\sum_{j=1}^{i}|X_j|$ for i=1,...,s. It can be shown that(see lemma 3.3 below) the maximum cardinality of any subset X_i is $\leq \frac{N}{s}\log N$, with probabilty greater than $1-O(N^{-\alpha})$ (for any α). This result immediately implies that $r_i \leq i\frac{N}{s}\log N$ with probability $> 1 - O(N^{-\alpha})$ thus proving our claim.

Lemma 3.3 A random $S \subseteq X$ of size s divides X into $s+1$ subsets as explained above. The maximum cardinality of any of the resulting subsets is $\leq \frac{\alpha}{2}\frac{N}{s}\log_e N$ with probability greater than $1-N^{-\alpha}$. $(|X|=N)$.

Proof

First we'll compute the probability that at least one subset is of size at least v. Realize that the selection of a random subset S of X results in a random $(s+1)$-partition of X. Total number of $(s+1)$ partitions of X is $\binom{N-1}{s}$. Number of partitions of X that have at least one part of size $\geq v$ is $\binom{N-v-1}{s}$.

Therefore, the probability that there is at least one part which is of size at least v is

$$P = \frac{\binom{N-v-1}{s}}{\binom{N-1}{s}}.$$

Using Stirling's approximation,

$$P \simeq \frac{(N-v-1)^{N-v-1}\,(N-s-1)^{N-s-1}}{(N-s-v-1)^{N-s-v-1}\,(N-1)^{N-1}}$$

$$\simeq \frac{\left(1-\frac{v+1}{N}\right)^{N-v-1}\left(1-\frac{s+1}{N}\right)^{N-s-1}}{\left(1-\frac{s+v+1}{N}\right)^{N-s-v-1}\left(1-\frac{1}{N}\right)^{N-1}}$$

Using the fact $\left(1-\frac{1}{w}\right)^w < \frac{1}{e}$,

$$P \leq \exp\left[\frac{-2vs}{N}\right]$$

If $v=\frac{\alpha}{2}\frac{N}{s}\log_e N$, then $P \leq N^{-\alpha}$.

i.e., the probability that none of the resulting subsets is of size greater than $\frac{\alpha}{2}\frac{N}{s}\log_e N$ is $> 1-N^{-\alpha}$, which is our claim.

This completes our proof of Lemmas 3.3 and 3.2. These Lemmas will be used repeatedly here after.

4. DERIVATION OF RANDOMIZED SELECT ALGORITHMS

4.1 A Summary of Select Algorithms

Let X be a set of N keys. We wish to derive efficient algorithms for select(i,X) where $1 \leq i \leq N$. Recall we wish to get the correct answer always but the run time may be a random variable. We display a canonical algorithm for this problem and then show how select algorithms in the literature follow as special cases of this canonical algorithm.

```
algorithm canselect;
  begin
      select a bracket (i.e., a subset) $B$ in $X$ such that
      select($i$,$X$) lies in this bracket with very high
      probability;
      Let $i_1$ be the number of keys in $X$ less than the
      smallest element in $B$;
      return canselect($i-i_1$ , $B$)
  end;
```

Select algorithm of [Hoare 61] chooses a random *splitter* key $k \in X$, and recursively considers either the low key set or the high key set based on where the ith element is located. And hence, B for this algorithm is either $\{x \in X \,/\, x \leq k\}$ or $\{x \in X \,/\, x > k\}$ depending on which set contains the ith largest element of X. $|B|$ for this algorithm is $\frac{N}{c}$ for some constant c.

On the other hand, select algorithm of [Floyd and Rivest 75] chooses two random splitters k_1 and k_2 and sets B to be $\{x \in X \,/\, k_1 \leq x \leq k_2\}$. k_1 and k_2 are chosen properly so as to make $|B| = O(N^{\alpha})$, $\alpha < 1$. We'll analyze these two algorithms in more detail now.

4.2 Hoare's Algorithm

Detailed version of Hoare's select algorithm is given below.

```
algorithm Hselect($i$,$X$);
                (assert $1 \leq i \leq |X|$)
   begin
      if $X = \{x\}$ then return $x$;
      Choose a random splitter $k \in X$;
      Let $B = \{x \in X \,/\, x < k\}$;
      if $|B| \geq i$ then return Hselect($i$,$B$)
      else return Hselect($i - |B|$ , $X - B$)
   end;
```

Let $T_p(i,N)$ be the expected parallel time of Hselect(i,X) using at most p simaltaneous comparisons at any time. Then the recursive definition of Hselect yields the following recurrence relation on $T_p(i,N)$.

$$T_p(i,N) = \frac{N}{p} + \frac{1}{N}\left[\sum_{j=1}^{i} T_p(i-j,\, N-j) + \sum_{j=i+1}^{N} T_p(i,j)\right].$$

An induction argument shows

$$T_N(i,N) \leq O(\log N)$$

and

$$T_1\,(i,N) \leq 2N + \min(i,N-i) + o(N).$$

To improve this Hselect algorithm, we can choose k such that B and $X - B$ are of approximately the same cardinality. This choice of k can be made by fusing samplerank$_s$ into Hselect as follows.

```
algorithm sampleselect$_s$ $(i,X)$;
            (assert $1 \leq i$ , $s \leq |X|$)
  begin
      if $X = \{x\}$ then return $x$;
      Choose a random sample set $S \subseteq X$ of size $s$;
      Let $k$ = select($\lfloor s/2 \rfloor$, $S$);
      Let $B = \{x \in X \ / \ x < k\}$;
      if $|B| \geq i$ then return sampleselect$_s$$(i,B)$
      else return sampleselect$_s$$(i - |B|\ , X - B)$
  end;
```

This algorithm can easily be analyzed using Lemma 3.2.

4.3 Algorithm of Floyd and Rivest

As was stated earlier, this algorithm chooses two keys k_1 and k_2 from X at random to make the size of its bracket $B=O(N^\beta)$, $\beta<1$. The actual algorithm is

```
algorithm FRselect$(i,X)$;
  begin
      if $X = \{x\}$ then return $x$;
      Choose $k_1$ ,$k_2 \in X$ such that $k_1 < k_2$ ;
      Let $r_1$ = rank($k_1$ , $X$) and $r_2$ = rank($k_2$ , $X$);
      if $r_1 > i$ then FRselect($i$,$\{x \in X \ / \ x < k_1\}$)
      else if $r_2 > i$ then FRselect($i-r_1$ , $\{x \in X \ / \ k_1 \leq x \leq k_2\}$)
         else FRselect($i-r_2$ , $\{x \in X \ / \ x > k_2\}$)
  end;
```

Let T_p (i,N) be the expected run time of the algorithm FRselect(i,X), allowing at most p simaltaneous comparisons at any time. Notice that we must choose k_1 and k_2 such that the case $r_1 \leq i \leq r_2$ occurs with high likelyhood and $r_2 - r_1$ is not too large. This is accomplished in FRselect as follows.

Choose a random sample $S \subseteq X$ of size s. Set k_1 to be select $\left(i\frac{(s+1)}{(N+1)} - \delta,\ S\right)$ and set k_2 to be select $\left(i\frac{(s+1)}{(N+1)} + \delta,\ S\right)$. If the parameter δ is fixed to be $\lceil\sqrt{d\alpha s \log N}\rceil$, for some constant d, then by lemma 3.2, Porb.$[r_1 > i] < N^{-\alpha}$ and Prob.$[r_2 < i] < N^{-\alpha}$. Let $T_p\,(-,s) = \max_j T_p\,(j,s)$. The resulting recurrence for the expected parallel run time with p processors is

$$\begin{aligned} T_p\,(i,N) &\le \frac{N}{p} + T_p\,(-,s) \\ &\quad + \text{Prob.}[r_1 > i] \times T_p\,(i,r_1) \\ &\quad + \text{Prob.}[i > r_2] \times T_p\,(i-r_1\,,\ N-r_2) \\ &\quad + \text{Prob.}[r_1 \le i \le r_2] \times T_p\,(i-r_1\,,\ r_2-r_1) \\ &\le \frac{N}{p} + T_p\,(-,s) + 2\,N^{-\alpha} \times N + T_p\left(i,\ \left\lceil \frac{N+1}{s+1}\delta \right\rceil\right). \end{aligned}$$

Note that k_1 and k_2 are chosen recursively. If we fix $d\alpha < 3$ and let $s = N^{2/3}\log N$, the above recurrence yields [Floyd and Rivest 75]

$$T_1\,(i,N) \le N + \min\,(i,N-i) + O(s).$$

Observe that if we have N^2 processors, we can solve the select problem in one time unit, since, all possible pairs of keys can be compared in one step. This implies that $T_p\,(i,N) = 1$ for $p \ge N^2$. Also, from the above recurrence relation,

$$T_N\,(i,N) \le O(1) + T_N\,(-,\sqrt{N}) = O(1),$$

as is shown in [Reischuk 81].

5. DERIVATION OF RANDOMIZED SORTING ALGORITHMS

5.1 A Canonical Sorting Algorithm

The problem is to sort a given set (X) of N distinct keys. The idea behind the canonical algorithm is to divide and conquer by splitting the given set into (say) s_1 disjoint subsets of almost equal cardinality, to sort each subset recursively, and finally merge the resultant lists. A detailed statement of the algorithm follows.

```
algorithm cansort($X$);
  begin
      if $X=\{x\}$ then return $X$;
      Choose a random subset $S \subseteq X$ of size $s$ ;
      Let $S_1$ be sorted $S$;
      As explained in section 3.2, $S_1$ divides $X$
      into $s+1$ subsets $X_1$ , $X_2$ , $\cdots$ , $X_{s+1}$ ;
      return  cansort($X_1$) . cansort($X_2$) . ... . cansort($X_{s+1}$);
  end;
```

Now we'll derive various sorting algorithms from the above.

5.2 Hoare's Sorting Algorithm

When $s = 1$ we get Hoare's algorithm. Hoare's sorting algorithm is very much similar to his select algorithm. Choose a random splitter $k \in X$ and recursively sort the set of keys $\{x \in X \ / \ x<k\}$ and $\{x \in X \ / \ x>k\}$.

```
algorithm quicksort($X$);
  begin
      if $|X| = 1$ then return $X$ ;
      Choose a random $k \in X$;
      return quicksort($\{x \in X \ / \ x<k\}$) . ($k$) . quicksort($\{x \in X \ / \ x>k\}$);
  end;
```

Let $T_1(N)$ be the number of sequential steps required by quicksort(X) if $|X| = N$. Then,

$$T_1(N) \leq N-1 + \frac{1}{N} \sum_{i=1}^{N} (T_1(i-1) + T_1(N-i)) \leq 2N\log N.$$

A better choice for k will be sampleselect$_s$ ($\lfloor N/2 \rfloor$, N). With this modification, quicksort becomes

```
algorithm samplesort$_s$ ($X$);
  begin
      if $|X| = 1$ then return $X$;
      Choose a random subset $S \subseteq X$ of size $s$;
      Let $k$ = select($\lfloor s/2 \rfloor$, $S$);
      return samplesort$_s$ ($\{x \in X \mid x < k\}$ . $(k)$ . samplesort$_s$ ($\{x \in X \mid x > k\}$);
  end;
```

By Lemma 3.2,

$$\text{Prob.}\left[|\text{rank}(k,X) - N/2| > \sqrt{d\alpha N}\log N \right] < N^{-\alpha}$$

for some constant d. Let $\overline{C}(s,N)$ be the expected number of comparisons required by samplesort$_s$ (X). Since the [Floyd and Rivest 75] selection algorithm requires only $N + o(N)$ comparisons, we have for $s(N) = N/\log N$,

$$\overline{C}(s(N), N) \leq \overline{C}(s(N_1), N_1) + N^{-\alpha}\,\overline{C}(s(N), N)N + o(N)$$

where

$$N_1 = N/2 + (\sqrt{d\alpha N})\log N.$$

Solving this recurrence [Frazer and McKeller 70] show

$$\overline{C}(s(N), N) \approx N\log_2 N\ ,$$

which asymptotically approaches the optimal number of comparisons required to sort N numbers on the comparison tree model.

Let $T_p(s,N)$ be the number of steps on a parallel comparison tree model with p processors to execute samplesort$_s$ (X) where $|X| = N$. Since only a constant number of steps are required to select the median k = select($N/2$, X) using N processors, [Reischuk 81] observes for this specialized algorithm with $s(N) = N$,

$$T_N(N,N) \leq O(1) + T_{N/2}(N/2, N/2)$$

$$\leq O(\log N).$$

5.3 Multiple Sorting

Any algorithm with $s>1$ falls under this category. Let us call cansort as multisort when $s>1$. As was shown in Lemma 3.3, the maximum cardinality of any subset X_i is $\leq \frac{N}{s}\log N$ $(= N_1, say)$ with probability $> 1 - O(N^{-\alpha})$. Therefore, if $T_p(N)$ is the expected parallel comparison time for executing multisort$_s$ (X) with p processors (where $|X| = N$), then,

$$T_p(N) \leq T_{pN_1/N}(N_1) + N^{-\alpha} T_p(N)$$

$$+ T_p(s) + \frac{N}{p} + \log(s)$$

$$\leq T_{pN_1/N}(N_1) + O(1) + \frac{N}{p}\log(s).$$

[Reischuk 82] uses the specialization $s = N^{1/2}$ which yields the following recurrence for $T_p(N)$.

$$T_N \leq T_{N_1}(N_1) + \frac{1}{2}\log N + O(1)$$

$$= O(\log N).$$

Alternatively we can set $p = N^{1+\epsilon}$ and $s = N^{\epsilon}$ for any $0 < \epsilon < 1$ and get an $N_1 = N^{1-\epsilon/2}\sqrt{d\alpha\log N}$ for some constant d. This choice of S yields the recurrence

$$T_{N^{1+\epsilon}}(N) \leq T_{N^{\epsilon}N_1}(N_1) + O(1) + N^{-\epsilon}\log N$$

$$\leq T_{N^{\epsilon}N_1}(N_1) + O(1)$$

$$\leq O(\log\log N)$$

5.4 FLASHSORT

[Reif and Valiant 83] give a method FLASHSORT for dividing X into even more equal sized subsets. This method is useful for sorts within fixed connection networks, where the processors can not be dynamically allocated to work on various size subsequences. The idea of [Reif and Valiant 83] is to choose a subsequence $S \subset X$ of size $N^{1/2}$, and then choose as splitters every ($\alpha \log N$)th element of S in sorted order, i.e., to choose $k_i' = \text{select}(\alpha i \lfloor \log N \rfloor, S)$ for $i = 1,2,\ldots,N^{1/2}/\log N$. Then they recursively sort each subset $X_i' = \{x \in X \ / \ k_{i-1}' < x < k_i'\}$. Their algorithm runs in time $O(\log N)$ and they have shown that after $O(\log N)$ recursive stages of their algorithm, the subsets will be of size not more than a factor of $O(1)$ of each other.

REFERENCES

[Aho, Hopcroft, and Ullman 76]

The Design and *Analysis of Algorithms*, Addison-Wesley Publications, 1976.

[Borodin and Hopcroft 82]

"Routing, Merging, and Sorting on Parallel Models of Computation", Proceedings of the 14th Annual ACM Symposium on Theory of Computing, 1982, pp 338-344.

[Fortune and Wylie 78]

Proceedings of the 10th Annual ACM Symposium on Theory of Computing, pp 114-118.

[Frazer and McKellar 70]

"Samplesort: A sampling Approach to Minimal Storage Tree Sorting", Journal of ACM, vol.17, No.3, July 1977, pp 496-502.

[Hoare 62]

"Quicksort", Computer Journal 5, 1962, pp 10-15.

[Floyd and Rivest 75]

"Expected Time Bounds for Selection", Communications of ACM, vol.18, March 1975, pp 165-172.

[Hoare 75]

"Algorithm 63 (PARTITION) and Algorithm 65 (FIND)", Communications of ACM, March 1975, vol.18, No.3.

[Kozen 81]

"Semantics of Probabilistic Programs", JCSS vol.22, 1981, pp 328-350.

[Knuth 73]

The Art of Compter Programming, vol.3, Sorting and Searching, Addison-

Wesley 1973.

[Meggido 82]

"Parallel Algorithms for Finding the Maximum and the Median Almost Surely in Constant Time", Preliminary Report, Compter Science Dept., Carnegie Mellon University, Pittsburg, PA, October 1982.

[Preparata 78]

"New Parallel Sorting Schemes", IEEE Transactions on Computers, vol. C27, No.7, July 1978, pp 669-773.

[Rabin 76]

"Probabilistic Algorithms", *Algorithms and Complexity, New Directions and Recent Results*, ed. by J. Traub, Academic Press 1976, pp 21-36.

[Reif and Scherlis 84]

"Deriving Efficient Graph Algorithms", Logics of Programs Workshop, Pittsburg, PA, 1984, Springer Verlag Notes in Computer Science 164.

[Reif and Valiant 83]

"A Logaritmic Time Sort for Linear Size Networks", 15th Annual ACM Symposium on Theory of Computing, Boston, MASS., 1983, pp 10-16.

[Reif 83]

"A $n^{1+\epsilon}$ Processor $O(\log\log n)$ Time Probabilistic Sorting Algorithm", SIAM Symposium on the Applications of Discrete Mathematics, Cambridge, MASS., June 27-29, 1983.

[Reischuk 81]

"A Fast Probabilistic Parallel Sorting Algorithm", IEEE Conference on Computer Science, Oct. 1981.

[Scherlis 80]

"Expression Procedures and Program Derivation", Ph.D Thesis, Stanford University 1980.

[Shiloach and Vishkin 81]

"Finding the Maximum, Merging, and Sorting in a Parallel Computation Model", Journal of Algorithms 2, 1981, pp 881-102.

[Valiant 75]

"Parallelism in Comparison Problems", SIAM Journal of Computing, vol.4, Sept. 1975, pp 348-355.

[Wilks 62]

Mathematical Statistics, John Wiley and Sons, New York, 1962.